VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Implementace systému pro správu záložních zdrojů napájení

Implementation of a System for Managment of Uninterruptible Power Supplies

Adam Bunček

Bakalářská práce

Vedoucí práce: Ing. Daniel Stříbný

Ostrava, 2021

Abstrakt

Tato práce je zaměřená na analýzu a vývoj systému pro správu záložních zdrojů napájení. Pro vyřešení dané problematiky bylo použito zařízení Raspberry Pi, na kterém běží systém pro monitoring jednotlivých zdrojů napájení. Systém je napsán v jazyce Python a naměřená data jsou pak zobrazována na webu, který je vytvořen pomocí web frameworku Django. Výsledkem je nezávislý systém, starající se o záložní zdroje, který umí pracovat autonomně.

Klíčová slova

Raspberry Pi; Python; Django; záložní zdroj napájení; web framework

Abstract

This work is focused on the analysis and development of a system for managing backup power supplies. To solve the problem, a Raspberry Pi device was used, on which a system for monitoring individual power supplies runs. The system is written in Python and the measured data are then displayed on a web, which is created using the Django web framework. The result is an independent system that takes care of backup resources, which can work autonomously.

Keywords

Raspberry Pi; Python; Django; backup power supply; web framework

Poděkování

Rád bych na tomto místě poděkoval Ing. Danielovi Stříbnému, který byl vždy na příjmu a byl ochotný mi s prací pomoct. Jeho optimismus a rady byly důležité pro zdárné dokončení této práce.

Obsah

Seznam použitých zkratek a symbolů

ADC	–	Analog-to-Digital Converter
NOOBS	–	New Out Of Box Software
UDP	–	User Datagram Protocol
UPS	–	Uninterruptible Power Supply/Source
PEMFC	–	Proton-exchange membrane fuel cells
MPPT	–	Maximum power point tracking
MOSFET	–	Metal Oxide Semiconductor Field Effect Transistor
GPIO	–	General-Purpose Input/Output
SoC	–	System-on-Chip
SD	–	Secure Digital
RAM	–	Random Access Memory
LAN	–	Local Area Network
PoE	–	Power over Ethernet
USB	–	Universal Serial Bus
API	–	Application Programming Interface
REST	–	Representational State Transfer
MVC	–	Model-View-Controller
HTTP	–	Hypertext Transfer Protocol

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódu

Kapitola 1

Úvod

V dnešní době již skoro vše funguje na elektřinu a žádná domácnost se bez ní neobejde dlouho. V případě výpadku dodávky elektrické energie může dojít k problému, proto není špatné mít po ruce něco jako náhradní zdroj energie. Pod pojmem náhradní zdroj energie si můžeme představit cokoliv, co nám vyrobí elektřinu, ať už se jedná o větrnou elektrárnu, solární panely, nebo třeba elektrocentrálu.

Právě myšlenka využití obnovitelných zdrojů jako záložního zdroje mě zaujala. Zájem o výrobu energie např. ze solárních panelů neustále roste, ale nesmíme zapomínat ani na neobnovitelné zdroje elektrické energie, které jsou pořád hojně používány. Každý z těchto zdrojů má své výhody či nevýhody, proto je třeba se při výběru záložního zdroje zamyslet nad jeho využitím.

Cílem této práce je navrhnout systém starající se o záložní zdroje, který bude pracovat nezávisle, bude monitorovat zdroje a předávat získané informace uživateli formou webového rozhraní. Uživatel však bude moci v případě potřeby zasáhnout a ovlivnit běh systému.

Celá práce se dá rozdělit do několika dílčích částí. V prvním kroku je zapotřebí zanalyzovat možné záložní zdroje a vybrat ty správné pro tuto práci. Po zvolení vhodných zdrojů přichází na řadu volba platformy, na které poběží zmiňovaný systém. Mezi další části patří návrh a implementace samotného systému. Posledním krokem pak je testování, kde se bude zkoušet chování systému na určité situace. Všechny tyto části, budou podrobně popsány v této práci.

Kapitola 2

Analýza problému

2.1 Možnosti záložního napájení

Každá budova je v dnešní době připojena do elektrické sítě. V momentě, kdy ale dojde k přerušení dodávky elektřiny nastane problém. Tato situace se nestává často, ale může k ní dojít prakticky kdykoliv. V takových situacích nám pak mohou posloužit záložní zdroje napájení.

Záložní zdroje napájení můžeme dělit podle několika kategorií:

1. Podle druhu výstupního napětí
    - Stejnosměrné
    - Střídavé
2. Podle druhu a způsobu přeměny
    - Rotační zdroje
    - Statické zdroje
    - Chemické zdroje

Mezi rotační zdroje většinou patří motorgenerátory, ve kterých dochází k přeměně paliva na kinetickou energii a následně na energii elektrickou. Mezi hlavní součásti tohoto systému patří spalovací motor a elektrický generátor (synchronní alternátor). V poslední době lze do kategorie rotačních zdrojů zařadit i setrvačníkový systém, kde je elektrická energie přeměněna na energii kinetickou. V tomto stavu je energie uložena a udržována do doby, než je nutné ji použít. Setrvačníkové systémy jsou konstruovány tak, aby bylo možné rychle měnit mezi generátorovým a spotřebičovým režimem. Statické záložní zdroje reprezentují tzv. UPS – Uninterruptible Power System (Nepřerušitelný zdroj napájení). Tyto zdroje fungují na principu uchování elektrické energie v akumulátorech. V případě použití dochází k přeměně uchované energie pomocí střídače, kde se stejnosměrné napětí přemění ve střídavé napětí. Statické zdroje dělíme do 3 kategorii:

1. Offline

2. Line-interactive

3. Online

Do kategorie chemických zdrojů spadá palivový článek. Palivové články jako záložní zdroj, se poslední dobou používají čím dál tím častěji. Existuje široká škála typů palivových článků, ale pro úlohu záložního zdroje se nejčastěji používá palivový článek s polymerním elektrolytem PEMFC (Polymer elektrolyte membrane fuel cell). [1]

2.2 Analýza vlastního řešení

Analýzu návrhu systému jsem rozdělil do dvou částí. Nejprve jsem se zaměřil na zdroje elektrické energie a jejich využití. Poté, na výběr zařízení, na kterém daný systém poběží.

Při výběru záložních zdrojů energie jsem se řídil situací, kdy mám chatu v horách, odříznutou od dodávky elektrické energie. Proto jsem jako jeden z prvních zdrojů vybral **12V akumulátor**. Tento typ zdroje zajistí, že systém poběží i bez nutnosti dodávky elektrické energie, ovšem jen na určitou dobu. Akumulátor je třeba neustále dobíjet, aby byl náš systém v provozu. Výdrž akumulátoru na jedno nabití závisí na jeho konstrukci. Vybral jsem olověný akumulátor pro jeho dostupnost a nízkou cenu.

Obrázek 2.1: 12V olověný akumulátor

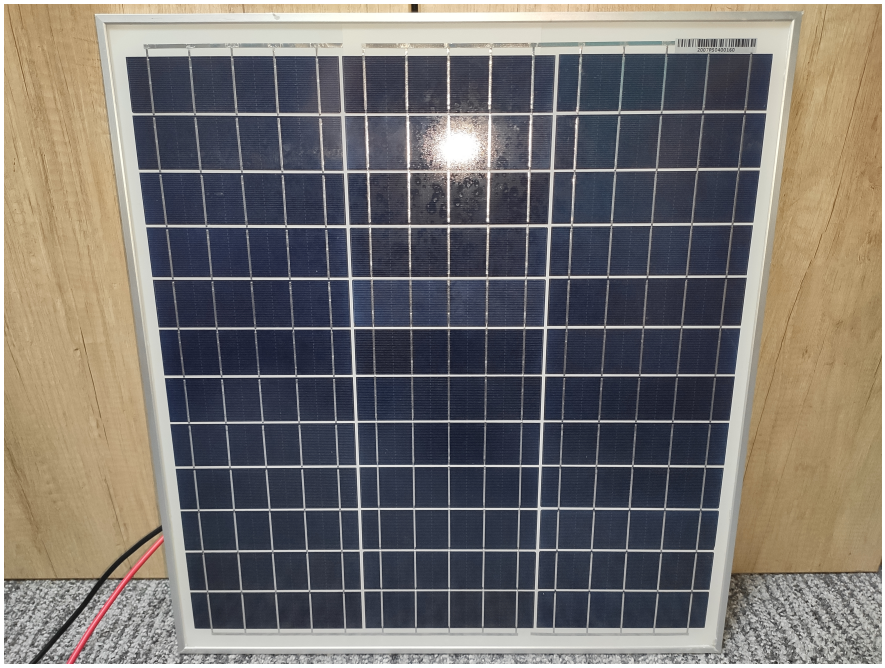
Ruku v ruce se s tímto zdrojem používají solární panely. Zájem o solární energii neustále roste a jedná se o velmi jednoduchý zdroj elektrické energie. Solární panely se postarají o dobíjení akumulátoru. Solární panely se dělí na 3 základní typy: **Monokrystalické**, **Polykrystalické** a **Amorfní**.

1. Monokrystalické mají nejvyšší účinnost co se týče přeměny sluneční energie na elektrickou. Pro dosažení této účinnosti, je ale třeba ideální osvětlení ve správném úhlu. Zároveň jsou náročnější na výrobu, což se promítá i do vyšší ceny.

2. Polykrystalické nabízí o něco menší účinnost, ale díky jejich struktuře lépe zachytí světlo přicházející z ostřejších úhlů. Jejich výkon je tak lépe rozložen v čase během dne.

3. Amorfní mají nejnižší účinnost. Jejich výhodou je však minimální tloušťka a hmotnost. Zároveň každý z nich může zachytit jinou vlnovou délku světla. [2]

Pro náš případ jsem zvolil **polykrystalický solární panel**, díky jeho vlastnostem se může akumulátor nabíjet po většinu dne.



Obrázek 2.2: Polykrystalický solární panel

Posledním vybraným zdrojem je **elektrocentrála**. Elektrocentrály se používají velice hojně jako záložní zdroj elektrické energie. Zejména na místech, kde si výpadek elektřiny může vyžádat ztrátu na životě nebo materiální škody. Elektrocentrála zajistí přísun elektrické energie tak dlouho, dokud jí nedojde palivo. Pomocí elektrocentrály pak můžeme snadno napájet spotřebiče na chatě.

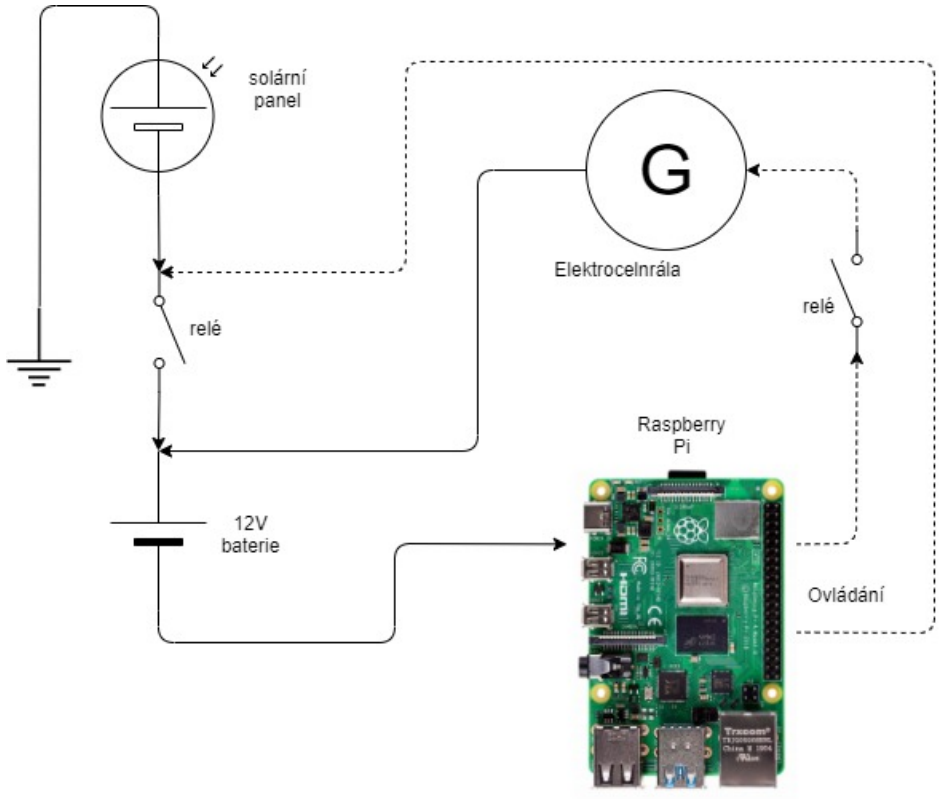
Samozřejmě, že zdrojů elektrické energie, které by se daly použít pro náš systém, je daleko více. Např. bychom mohli použít jednoduchou větrnou turbínu.

Druhou výše jmenovanou částí byl výběr zařízení. Zde bylo na výběr ze dvou možností. Arduino nebo Raspberry Pi. Arduino je levnější, lze jej napájet z baterie a je velmi jednoduché k němu připojit široké spektrum senzorů a další elektroniky. Navzdory tomu jsem však vybral Raspberry Pi. Proč? Arduino je totiž mikrokontroler, na kterém může běžet pouze jeden program stále dokola. Narozdíl od Arduina je Raspberry Pi jednodeskový počítač, na kterém může běžet několik programů současně. Raspberry Pi se dá rovněž napájet z baterky a pro práci se senzory je možné použít nepřeberné množství knihoven. Doporučený programovací jazyk pro Raspberry Pi je Python, což je další výhoda. Python má totiž jednoduchou syntaxi a je vhodný jak pro psaní krátkých jednoduchých programů, tak i pro velké aplikace.

Kapitola 3

Návrh vlastního řešení

Po důkladné analýze jsem pro tuto práci zvolil 3 typy zdrojů, konkrétně to jsou: 12V akumulátor, 40W solární panel a elektrocentrála. Tyto zdroje bude monitorovat a spravovat systém napsaný v jazyce Python, běžící na zařízení Raspberry Pi. Systém bude monitorovat napětí na akumulátoru, panelu i elektrocentrále. Dále bude monitorovat okolní teplotu a hladinu paliva v nádrži elektrocentrály.

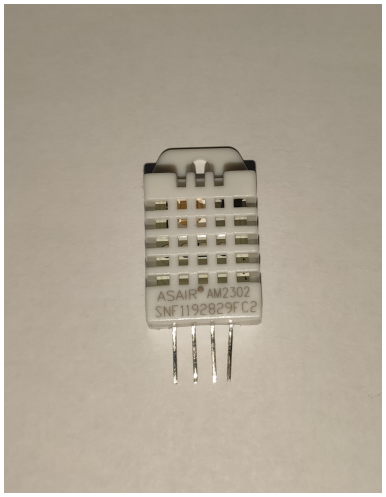


Obrázek 3.1: Schéma zapojení

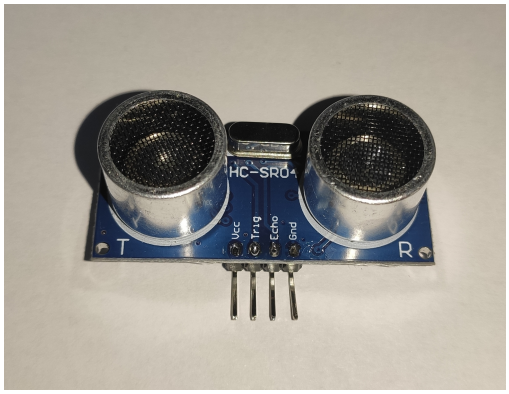
Pro práci byly použity dvě zařízení Raspberry Pi 4B, jedno v režimu slave a druhé v režimu master. Slave zařízení se stará o hlavní monitoring a posílá data na zařízení master. Na zařízení master běží webové rozhraní, kde uživatel může vidět aktuálně získané informace a bude mít možnost zvolení profilu, podle kterého se bude systém chovat. Systém bude pracovat nezávisle, ale v případě potřeby může uživatel zasáhnout a ovlivnit chod celého systému.

3.1 Hardware

Kromě zařízení Raspberry Pi bylo taky zapotřebí AD převodníku, ultrazvukového senzoru pro zjištění vzdálenosti, teplotního čidla, MPPT regulátoru a MOSFET relé.



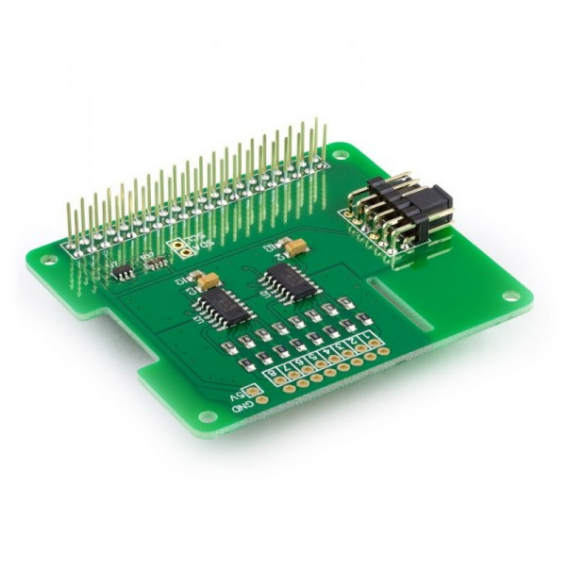
Obrázek 3.2: Teplotní čidlo DHT22



Obrázek 3.3: Ultrazvukový měřič vzdálenosti HC-SR04

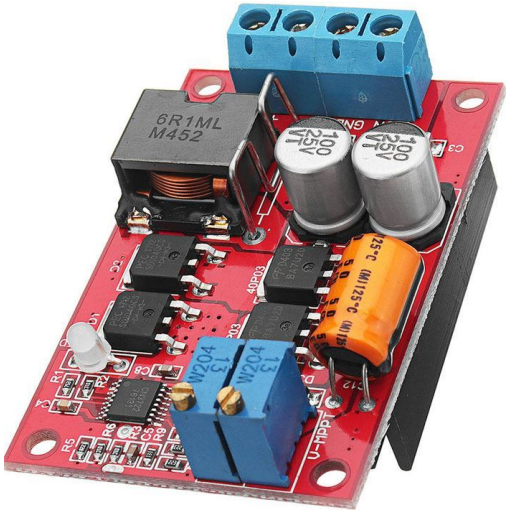
ADC Pi Plus je osmi kanálový sedmnácti bitový převodník analogového signálu na digitální. Převodník slouží pro monitoring napětí jednotlivých zdrojů. ADC Pi Plus používá odporové děliče

napětí na každém vstupu, pro snížení vstupní hodnoty. Vstupní napětí nesmí přesáhnout 5,06V. Proto, abychom naměřili realné hodnoty potřebujeme přidat zátěž do obvodu a tuto naměřenou hodnotu pak vynásobit konstantou pro daný odpor. [3]



Obrázek 3.4: ADC Pi Plus

MPPT-V08A je regulátor pro solární systémy s výstupním napětím 9-24V. Regulátor obsahuje měnič, který upraví V-A charakteristiku napájecího zdroje na úroveň, která je nejvhodnější pro nabíjení připojené baterie. Regulátor má dva trimry, kterými se nastavují výstupní hodnoty. [4]



Obrázek 3.5: regulátor MPPT-V08A

3.1.1 Raspberry Pi

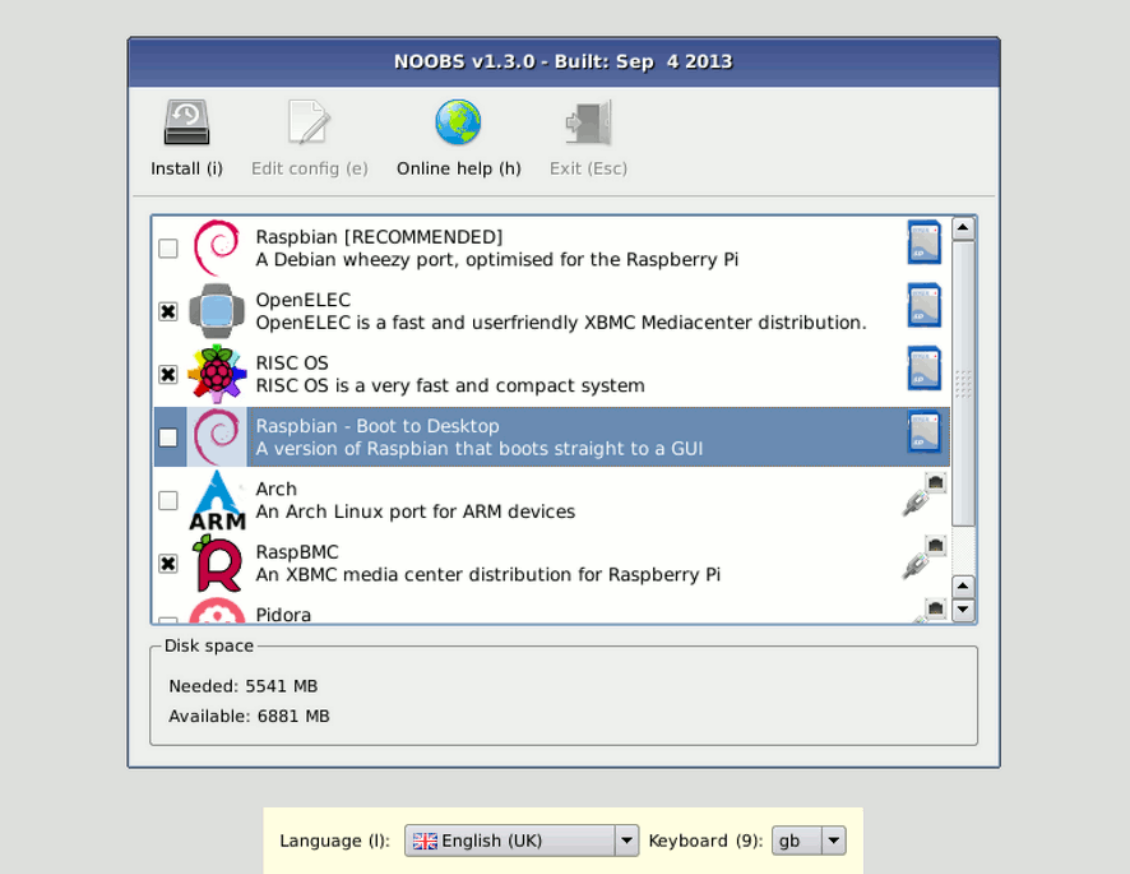
Raspberry Pi je malý jednodeskový počítač o velikosti zhruba platební karty. Je taky levnější než klasický počítač a můžeme u něj využít přístup k GPIO pinům, což mnoho zařízení nemá. Většina zařízení je dodávána pro konkrétní účel (herní konzole pro sledování filmů nebo hraní her, notebook pro práci atd.). Raspberry Pi se může stát kterýmkoliv z těchto zařízení. [5]

Jádrem systému Raspberry Pi je multimediální procesor typu SoC (system-on-chip) Broadcom BCM2835. Tento procesor se od jiných neliší jen svým návrhem, ale používá také jinou architekturu instrukční sady, která se označuje jako ARM. Architektura ARM, kterou již v 80. letech vyvinula společnost Acron Computers, se v počítačích uplatňuje poměrně zřídka. Vyniká však v mobilních zařízeních. Což vysvětluje, jak je možné, že počítač Raspberry Pi dokáže fungovat se zdrojem napájení s napětím jen 5V a proudem 1A. Zároveň to však znamená, že počítač Raspberry Pi není kompatibilní s tradičním softwarem pro PC. Většina programů pro stolní počítače a notebooky totiž odpovídá architektuře s instrukční sadou x86 nebo x64. Dalším rozdílem mezi Raspberry Pi a stolním počítačem je operační systém. Většina stolních i přenosných PC v současnosti používá operační systémy Microsoft Windows nebo Apple OS X. Obě tyto platformy spadají do kategorie closed source a vznikají v nepřístupném prostředí. Návrh počítače Raspberry Pi počítá s tím, že na něm bude fungovat operační systém GNU/Linux. Linux je systém typu open source. Díky tomu si můžete stáhnout zdrojový kód systému a změnit v něm cokoliv chcete. Proto bylo možné rychle přizpůsobit systém Linux pro počítač Raspberry Pi. Existuje několik verzí systému Linux, které se nazývají distribuce. Patří k nim např.: Debian, Fedora Remix a Arch Linux. [6]

Jako primární operační systém pro Raspberry Pi byl zvolen Raspbian. Raspbian je bezplatný operační systém založený na Debianu, optimalizovaném pro hardware Raspberry Pi. Raspbian však poskytuje více než čistý operační systém. Je dodáván s více než 35 000 předkompilovanými balíčky ve formátu pro snadnou instalaci. Raspbian je stále v aktivním vývoji. [7]

NOOBS je pomůcka pro instalaci operačního systému. Výhodou je možnost nainstalovat víc systémů na jednu SD kartu. Při startu je potom možné vybrat si jaký systém se má spustit. Stáhnutý .zip soubor obsahující NOOBS rozbalíme do předem zformátované SD karty. Poté SD kartu vložíme do našeho Raspberry Pi. Po spuštění se zobrazí tabulka se seznamem operačních systémů. U každého je vidět jestli pro instalaci potřebuje připojení k internetu nebo jestli se systém nainstaluje z SD karty. Můžeme si zaškrtnout kolik systému chceme, tak aby se nám na SD kartu vešly. Jakmile skončí instalace objeví se podobná tabulka se seznamem nainstalovaných systému. Ta se objeví i při každém spuštění Raspberry Pi. Stačí si tedy vybrat, který systém se má spustit. [8]

Raspberry Pi má řadu portů, ale těmi nejdůležitějšími pro tuto práci jsou GPIO piny. Hlavička GPIO se skládá ze 40 pinů ve 2 řadách. Existuje několik kategorií pinů, z nichž každá má určitou funkci. Mezi další důležité porty na Raspberry Pi patří slot pro SD kartu, na které je nainstalován operační systém. [9]



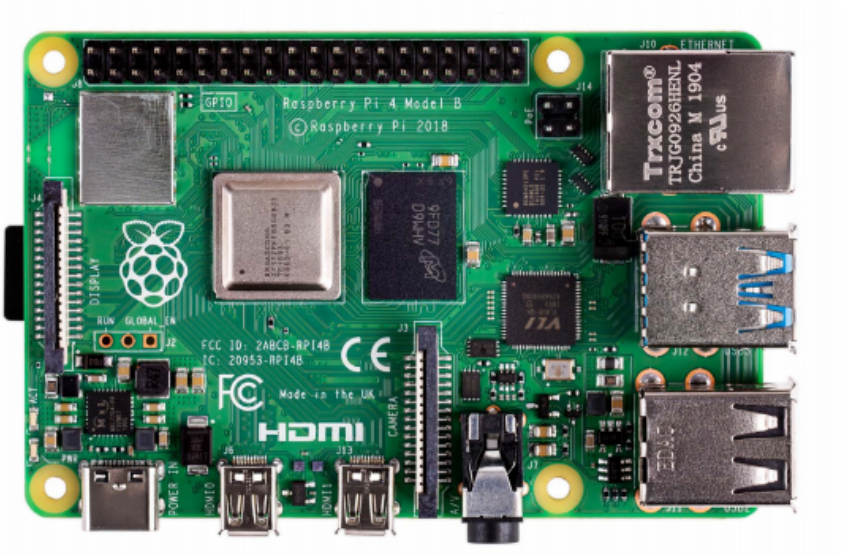
Obrázek 3.6: NOOBS
[8]

Kategorie	Popis
3V3	trvale zapnutý 3,3V zdroj
5V	trvale zapnutý 5V zdroj
Ground(GND)	uzemnění, pro uzavření obvodu
GPIO XX	GPIO piny pro komunikaci s připojeným hardwarem označené číslem od 2 do 27
ID EEPROM	piny vyhrazené pro připojení dalšího hardwaru (HAT) nebo dalšího příslušenství

Tabulka 3.1: Kategorie GPIO pinů

3.1.1.1 Raspberry Pi 4B

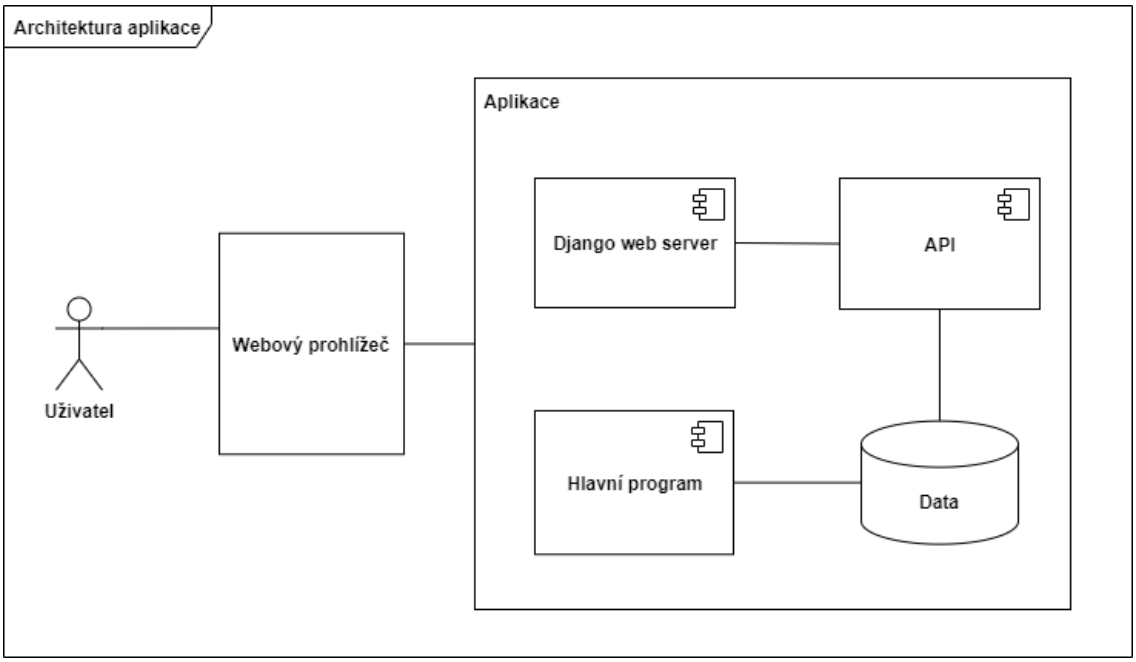
Raspberry Pi 4 Model B je nejnovější produkt Raspberry Pi počítačů. Nabízí zvýšení rychlosti procesoru, multimediálního výkonu, paměti a konektivity oproti předchozím modelům. Zároveň zachovává zpětnou kompatibilitu a podobnou spotřebu energie. Raspberry Pi 4 Model B poskytuje uživateli výkon srovnatelný s ostatními PC systémy. Mezi klíčové funkce tohoto produktu patří vysoce výkonný 64 bitový čtyřjádrový procesor, podpora dvou displejů v rozlišení až 4K, hardwarové dekódování videa až do 4Kp60, až 8 GB RAM, dvoupásmová bezdrátová síť LAN 2,4 / 5,0 GHz, Bluetooth 5.0, Gigabit Ethernet, USB 3.0, a schopnost PoE (prostřednictvím samostatného doplňku PoE HAT). [10]



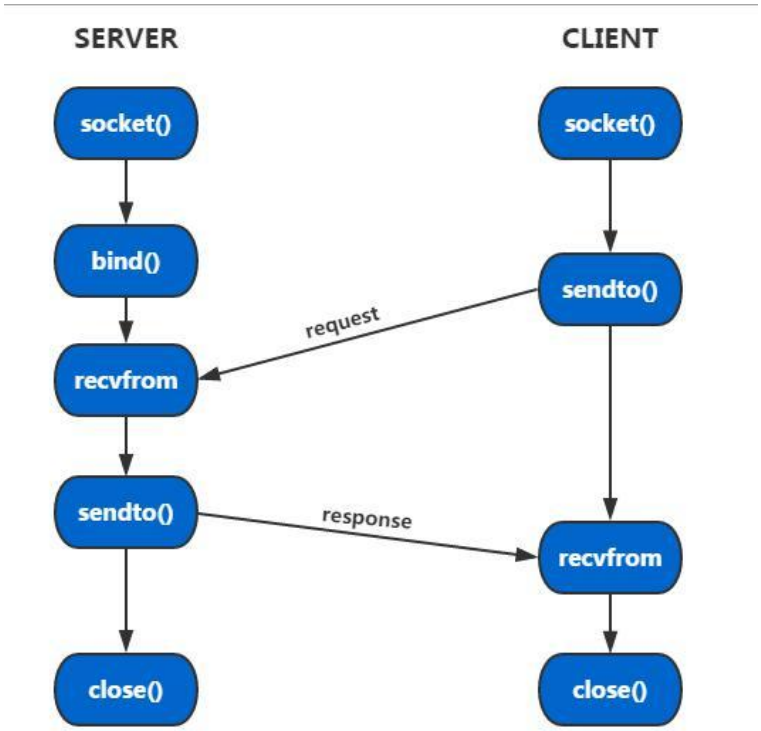
Obrázek 3.7: Raspberry Pi 4 Model B
[10]

3.2 Software

Programová část práce je napsána v jazyce Python, webové prostředí je pak vytvořeno pomocí web frameworku Django. Na slave zařízení běží hlavní monitorovací program, který se také stará o vykonávání přednastavených profilů. Veškeré nashromážděné infromace pak toto zařízení pomocí UDP paketů posílá na zařízení master. Zde běží program, který tyto pakety odchytává. Pro komunikaci mezi programem a webovým rozhraním slouží RESTové API. Pro samostatný chod systému jsou přednastaveny dva profily. Uživatel si může tyto profily upravit podle vlastních potřeb. Konkrétně si může určit čas, kdy se zapne nebo vypne nabíjení ze solárních panelů, při jaké hladině paliva vypnout elektrocentrálu, aby nedošlo k jejímu poškození a při jak nízkém stavu baterie zapnout elektrocentrálu pro její dobití.



Obrázek 3.8: Architektura aplikace



Obrázek 3.9: Komunikace pomocí UDP paketů
[11]

3.2.1 Python

Python vznikl koncem 80. let v institutu National Research Institute for Mathematics and Computer Science a jeho autor Guido van Rossum vycházel z jazyka ABC. Od svého začátku jazyk Python neustále získává na oblibě díky jasné a expresivní syntaxi, při jejímž vývoji byl kladen důraz na čitelnost kódu. Python je vysokoúrovňový jazyk. To znamená, že kód jazyka Python se skládá ze srozumitelných anglických slov. [6]

Kód Pythonu je obvykle třikrát menší, než stejný kód v C ++ nebo Javě. Programy v Pythonu také běží okamžitě, bez zdlouhavých kroků kompilace. Většina programů Pythonu běží beze změny na všech hlavních počítačových platformách. Přenesení kódu Pythonu například mezi Linuxem a Windowsem je obvykle jen otázkou kopírování zdrojového kódu mezi platformami. Python navíc nabízí více možností pro kódování přenosných grafických uživatelských rozhraní, programů pro přístup k databázím, webových systémů a dalších. Python přichází s velkou sbírkou předem připravených funkcí, známých jako standardní knihovna. Tato knihovna podporuje řadu úloh na úrovni aplikace, od shody textových vzorů až po skriptování v síti. Python lze navíc rozšířit o rozsáhlou sbírku softwaru pro podporu aplikací třetích stran. Doména třetí strany Pythonu nabízí například nástroje pro konstrukci webových stránek, numerické programování, přístup k sériovému portu, vývoj her a mnoho dalšího. Z hlediska funkcí je Python hybridní jazyk. Jeho sada nástrojů jej umisťuje mezi tradiční skriptovací jazyky (například Tcl, Scheme a Perl) a vývojové jazyky (jako C, C ++ a Java). Python poskytuje veškerou jednoduchost a snadné použití skriptovacího jazyka spolu s pokročilejším softwarovým inženýrstvím. Díky této kombinaci je Python užitečný pro široké spektrum projektů. [12]

3.2.2 Django

Django je open source webový framework v Pythonu, který podporuje rychlý vývoj a čistý, pragmatický design. Django byl navržen tak, aby pomohl vývojářům co nejrychleji přenést aplikaci z fáze konceptu až do její finální podoby. [13]

Django využívá architekturu Model-View-Controller. Jádro frameworku obsahuje objektově-relační mapper, který je prostředníkem mezi datovým modelem a relační databází, systémem zobrazení pro zpracování požadavků a šablonovacím systémem. [14]

Django je napsáno v čistém Pythonu. Pro použití tedy vyžaduje instalaci Pythonu 2.3 nebo vyšší. Django obsahuje vestavěný webový server. Při vývoji tak nemusíme řešit konfiguraci webového serveru (např. Apache). Tento server sleduje změny kódu a automaticky se znovu načte, což umožňuje provést změny na projektu, aniž by bylo třeba cokoli restartovat. Pro spuštění serveru musíme spustit příkaz *python manage.py runserver*. Webový server pak bude dostupný na stránce *http://127.0.0.1:8000/*. [15]

Kapitola 4

Implementace

4.1 Zařízení slave

Hlavní monitorovací program běžící na zařízení slave se skládá z několika modulů, které hlavní program volá. Jsou to moduly pro jednotlivá měření, monitoring a modul pro vykonání předem nastavených profilů. Mezi moduly pro monitoring patří modul pro zjištění hladiny paliva v elektrocentrále, modul pro získání teploty z teplotního čidla a modul pro měření napětí na jednotlivých zdrojích.

V modulu pro získání hladiny paliva je třeba nejdříve nastavit GPIO piny pro vstup a výstup. Dále probíhá měření vzdálenosti. Jelikož se jedná o ultrazvukový senzor, jeho princip je jednoduchý. Jakmile vyšle signál, začne se měřit čas. Až se signál odrazí od hladiny a vrátí se zpět, čas se stopne. Dále již následuje výpočet vzdálenosti od hladiny. V případě, že se nepodaří naměřit žádná data, uloží se o tom infromace do logu a běh systému se přeruší.

```
import RPi.GPIO as GPIO
import time
import log
import sys
import myGlobals
GPIO.setmode(GPIO.BCM)

#funkce pro ziskani vzdalenosti od hladiny paliva v nadrzi elektrocentraly
def getFuelLevel(trig, echo):

    #nastaveni vstupniho a vystupniho pinu
    GPIO.setup(trig, GPIO.OUT)
    GPIO.setup(echo, GPIO.IN)
```

```
GPIO.output(trig, False)

try:
    GPIO.output(trig, True)
    time.sleep(0.00001)
    GPIO.output(trig, False)

    #vyslani signalu
    while GPIO.input(echo)==0:
        pulse_start = time.time()

    #ziskani odezvy
    while GPIO.input(echo)==1:
        pulse_stop = time.time()

    #vypocet vzdalenosti
    pulse_time = pulse_stop - pulse_start
    #podle vzorce v = s/t
    distance = pulse_time * 17150

    distance = round(distance,2)
    log.logger.info('Hladina paliva je {}, zjisteno na pinu {}'.format(
        distance, echo))

except Exception:
    log.logger.exception('Nebyla namerena zadna hodnota v palivove nadrzi.
        Zkontrolujte zapojeni. Pin cislo: {}'.format(echo))
    GPIO.output(myGlobals.GENRELAY, False)
    GPIO.output(myGlobals.PANRELAY, False)
    sys.exit()

else:
    return distance
```

Ukázka zdrojového kódu 4.1: Modul pro získání hladiny paliva

Modul pro získání teploty využívá knihovnu *Adafruit_DHT*. Opět musíme nastavit potřebné GPIO piny a pak stačí zavolat funkci knihovny, která nám vrátí naměřené hodnoty. V případě chyby se uloží chybové hlášení do logu a aplikace se ukončí.

```
import Adafruit_DHT as dht
import RPi.GPIO as GPIO
import log
import myGlobals
import sys

#nastaveni pinu
GPIO.setmode(GPIO.BCM)

DHT_SENSOR = dht.DHT22

#funkce pro ziskani teploty
def getTempInfo(pin):
    try:
        hum, temp = dht.read_retry(DHT_SENSOR, pin)
        log.logger.info('Namerena teplota {} na pinu {}'.format(round(temp,2), pin
            ))

    except:
        log.logger.exception('Nebyla namerena zadna hodnota na teplomeru.
            Zkontrolujte zapojeni. Pin cislo: {}'.format(pin))
        GPIO.output(myGlobals.GENRELAY, False)
        GPIO.output(myGlobals.PANRELAY, False)
        sys.exit()

    else:
        return (round(temp,2),hum)
```

Ukázka zdrojového kódu 4.2: Modul pro získání teploty

Modul pro měření napětí na zdrojích používá knihovnu *ADCPi*. Výše zmíněný AD převodník má 8 kanálů pro převod analogového signálu na digitální. Těchto 8 kanálů jsem rozdělil pro naše 3 zdroje. První 3 jsou pro baterie, kanály 4 a 5 jsou vyhrazeny pro elektrocentrály a poslední 3 jsou ponechány pro solární panely. Ve funkci se prochází dané rozpětí a kontroluje se, zda je na daném kanálu nějaké napětí. Pokud ano, tak se toto napětí uloží do předem připravených proměnných. Když se nepodaří nic naměřit, uloží se chybové hlášení do logu. V ukázce kódu je pouze funkce pro baterie, ale funkce pro elektrocentrály a panely jsou kromě rozpětí kanálů totožné.

```
from ADCPi import ADCPi
import log
adc = ADCPi(0x68, 0x69, 18)

#listy pro pripad vice pripojenych zdroju
batteries = []
generators = []
panels = []

MULTIPLIER = 2.5 # nasobitel pro maximalni napeti 12.7V

# 8 kanalu pro zjisteni napeti 1-3 baterky, 4-5 generatory, 6-8 panely

#funkce pro mereni napeti na baterkach
def batteriesInfo():
    try:
        for i in range(3):
            #zavolani funkce knihovny
            if adc.read_voltage(i+1) > 1.0:
                #kdyz se na danem kanalu objevi napeti, vynasobi se a zaokrouhli
                vol = round(adc.read_voltage(i+1)*MULTIPLIER,2)
                batteries.append(vol)
                log.logger.info('Namereno napeti {} na kanale {}'.format(vol, i+1))
        return (batteries)
    except:
        log.logger.exception('Nebyla namerena zadna hodnota na bateriich.
            Zkontrolujte zapojeni. Kanal cislo: {}'.format(i+1))
        return(0)

    batteries.clear()
...
```

Ukázka zdrojového kódu 4.3: Modul měření napětí na zdrojích

V modulu pro vykonávání aktivního profilu se kontrolují naměřené hodnoty s hodnotami v profilu. Profil si může uživatel nastavit podle sebe pomocí webového rozhraní. Kontroluje se stav hladiny paliva, napětí na bateriích a čas během kterého se má baterie nabíjet z panelu. V ukázce kódu je pouze kontrola napětí baterie a kontrola času pro nabíjení ze solárního panelu.

```
...

#12.7V a vic je 100% kapacity baterie
if b < 12.7:
    #kontrola napeti baterie
    if b < float(profile["minvoltage"] or (b > float(profile["minvoltage"]) and b
        < 12.7)):
        #kontrola casu pro zapnuti nabijeni z panelu
        if timeCheck(datetime.now().time(), datetime.strptime(profile["timefrom"],
             "%H:%M:%S").time(), datetime.strptime(profile["timeto"], "%H:%M:%S").
            time()):
            GPIO.output(myGlobals.PANRELAY, True)
            log.logger.info('Baterie se nabiji z panelu. Napeti na baterii: {}'.
                format(b))
        else:
            GPIO.output(myGlobals.PANRELAY, False)
            log.logger.info('Panel je odpojen. Napeti na baterii: {}'.format(b))
else:
    GPIO.output(myGlobals.PANRELAY, False)
    log.logger.info('Panel je odpojen. Napeti na baterii: {}'.format(b))

...
```

Ukázka zdrojového kódu 4.4: Vykonávání aktivního profilu

V hlavním programu dále probíhá neustálá komunikace se zařízením master. Obě zařízení neustále naslouchají a přijímají zprávy jeden od druhého. V jazyce Python pro tuto komunikaci slouží knihovna *socket*.

```
import socket

#ip zarizeni master
UDP_IP = "192.168.0.107"
UDP_PORT = 12345

#vytvoreni socketu pro komunikaci, SOCK_DGRAM = UDP
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_address = (UDP_IP, UDP_PORT)
```

```
#zprava s informacemi pro zarizeni master
message = {"dateTime": now.strftime("%d.%m.%Y %H:%M:%S"),
"fuelLevels": fuelLevels, "batteryVoltages": batteryVoltages,
"genVoltages": genVoltages, "panelVoltages": panelVoltages,
"temperature": tempAndHum[0], "relays" : relays, "infos":infos,
"profile": profile["id"], "ai": user["ai"]}

# poslani zpravy na master
sent = sock.sendto(message, server_address)

# cekani na odpoved - 5 vterin
sock.settimeout(5)
log.logger.info('Cekani na odpoved')

#ocekava se prichod zpravy o maximalni delce 512b
data, server = sock.recvfrom(512)
```

Ukázka zdrojového kódu 4.5: Komunikace pomoci UDP paketů

4.2 Zařízení master

Samotné zařízení master je rovněž uzpůsobeno pro monitoring zdrojů pro případ, že každé zařízení bude mít své vlastní zdroje. V tomto případě, jsou ale obě zařízení připojená ke stejným zdrojům. Zařízení master přijímá zprávy ze zařízení slave a ukládá je do souboru. Tato data jsou následně zobrazována na webovém rozhraní pomocí jednoduchého REST API.

4.2.1 REST API

API (Application Programming Interface) slouží pro komunikaci s počítačem nebo systémem za účelem načtení informací nebo provedení určité funkce.

REST (Representational State Transfer) umožňuje jednoduše číst, editovat nebo mazat informace ze serveru pomocí jednoduchých HTTP příkazů.

V jazyce Python existují knihovny *flask* a *flask_restful* pro vytvoření tohoto API. V ukázce kódu můžeme vidět třídu *SlaveData*, která má definovanou metodu *get*. V metodě se neděje nic jiného než načtení dat ze souboru. Při každém zavolání metody *get* se nám tedy vrátí obsah souboru s aktuálními daty. Dále následuje příkaz *api.add_resource*, který vytvoří odkaz na webu pro třídu *SlaveData*.

```
from flask import Flask, request
from flask_restful import Api, Resource
import fileManagement as file
import log

app = Flask(__name__)
api = Api(app)

#trida pro ziskani namerenych dat
class SlaveData(Resource):
    #metoda get pro ziskani dat ze souboru
    def get(self):
        log.logger.info('GET SlaveData')
        return (file.readData("data.json"))

api.add_resource(SlaveData, "/slave")
```

Ukázka zdrojového kódu 4.6: REST API

Při zadání adresy *http://127.0.0.1:5000/slave* dostaneme následující výstup. Na obrázku můžeme vidět data, která byla poslána ze zařízení slave na zařízení master. Tato data byla uložena do souboru a následně přečtena pomocí metody *get*.

Obrázek 4.1: REST API

4.2.2 Web

Jak už bylo zmíněno dříve, webové rozhraní je vytvořeno pomocí webframeworku *Django*. Protože je Django založen na architektuře MVC, hlavní práce se odehrávala především v modulu *views.py*. Každá stránka aplikace je reprezentována funkcí v tomto modulu. Jak je vidět v ukázce kódu, hlavní stránka je reprezntována funkcí *index*. Na začátku této funkce je snaha získat právě aktivní profil za využití našeho API. Funkce *get* vrátí infromaci o tom, který z profilů je právě aktivní. Podle toho se pak nastaví výchozí hodnota formuláře. Dále se kontroluje POST request. Jestliže byl zachycen, došlo nejspíš ke změně aktivního profilu. Proto data získaná z POST requestu putují na API a následně se tyto informace kontrolují v hlavnímu programu na zařízení master. Takto funguje i zapínaní nebo vypínání elektrocentrály uživatelem.

```
#hlavni stranka aplikace, volba profilu
def index(request):

    try:
        form = ProfilesForm()
        #ziskani dat
        profile = requests.get('http://127.0.0.1:5000/profiles')
```

```
        profile = json.loads(profile.text)

        activeProfileId = profile["id"]
        #nastaveni vychozich hodnot pro formular
        form.fields['profile_name'].initial = profile["id"]
        form.fields['profile_name'].disabled = False

        #kontrola POST requestu
        if request.method == 'POST':
            form = ProfilesForm(request.POST)
            if form.is_valid():
                #ziskani dat z POSTu
                profileId = form.cleaned_data['profile_name']
                activeProfileId = profileId

                form.fields['profile_name'].initial = [profileId]
                #odeslani dat na API
                postedData = {"profile" : profileId}
                requests.post('http://127.0.0.1:5000/profiles', postedData)
        #nacteni templatu
        data["form"] = form
        data["activeProfileId"] = activeProfileId
        return render(request, 'index.html', data)
    except:
        #spojeni s api nebylo navazano
        print("\nConnection with api not established")
        #nacteni templatu
        data["form"] = "Pripojeni k Api nebylo navazano"
        return render(request, 'index.html', data)
```

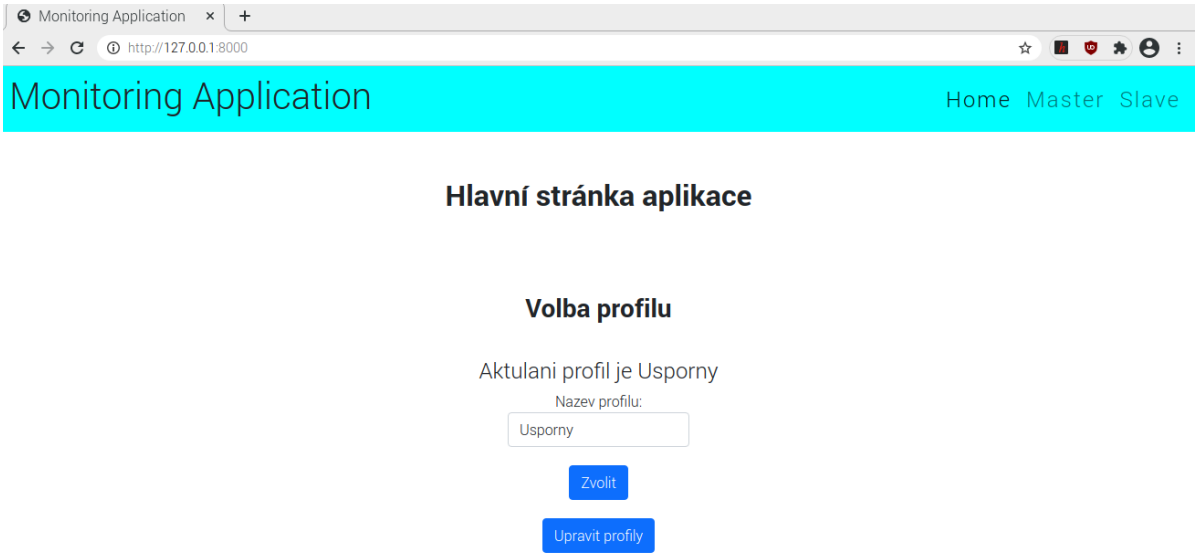
Ukázka zdrojového kódu 4.7: Hlavní stránka

Zde probíhá kontrola aktivního profilu s informacemi ze zařízení slave. V momentě, kdy se změní aktivní profil na zařízení master, API uloží tuto informaci do konfiguračního souboru *profiles.ini*. Když potom hodnoty ze zařízení slave nesouhlasí s hodnotami ze zařízení master, odešle se o tom zpráva zpátky na slave a profil se nastaví tak, aby souhlasil s tím co je uloženo v konfiguračním souboru na masterovi.

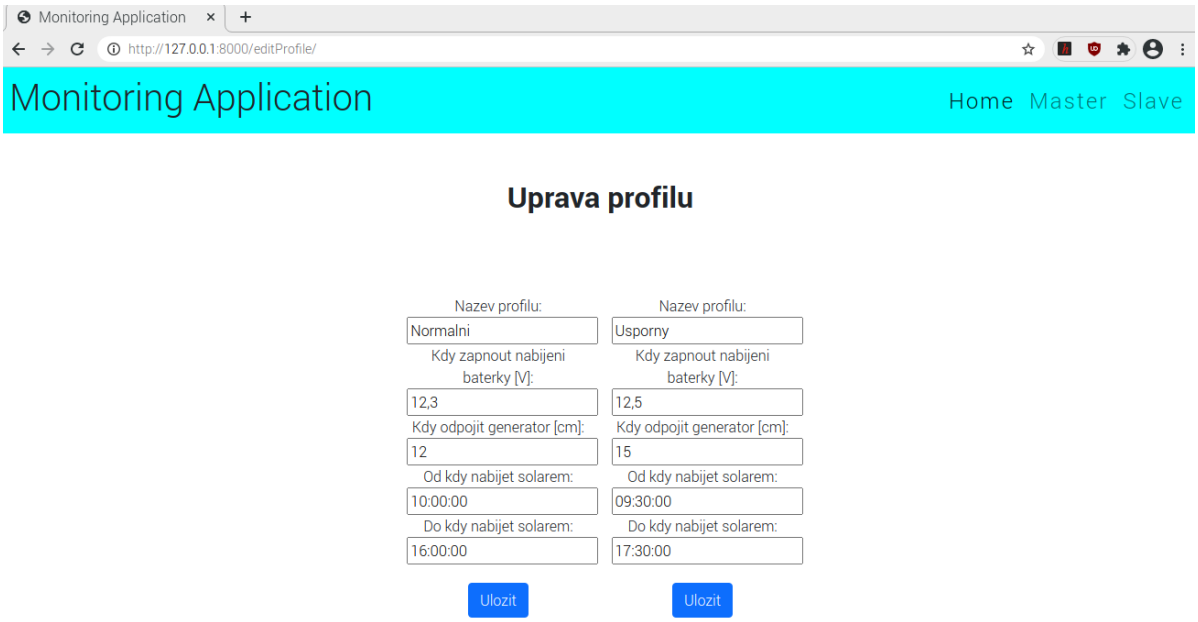
```
#kontrola aktualniho profilu
    actualProfile = file.parseProfile()

    if slaveData["profile"] != actualProfile["id"]:
        #kdyz hodnoty nesedi, odesle se o tom informace, na jejimz zaklade se bud
            zapne nebo vypne rele
        if profileSent == False:
            sent = sock.sendto(json.dumps(actualProfile).encode("utf-8"), address)
            print('sent {} back to {}'.format(actualProfile["name"], address))
            profileSent = True
    else:
        profileSent = False
        print("equals")
```

Ukázka zdrojového kódu 4.8: Kontrola aktivního profilu

Design webu je jednoduchý a přehledný. Na hlavní stránce se nachází volba profilu podle kterého systém poběží. Uživatel může tento profil kdykoliv přepnout nebo upravit podle svého uvážení. Na výběr je ze 2 profilů: *Normální* a *Úsporný*.

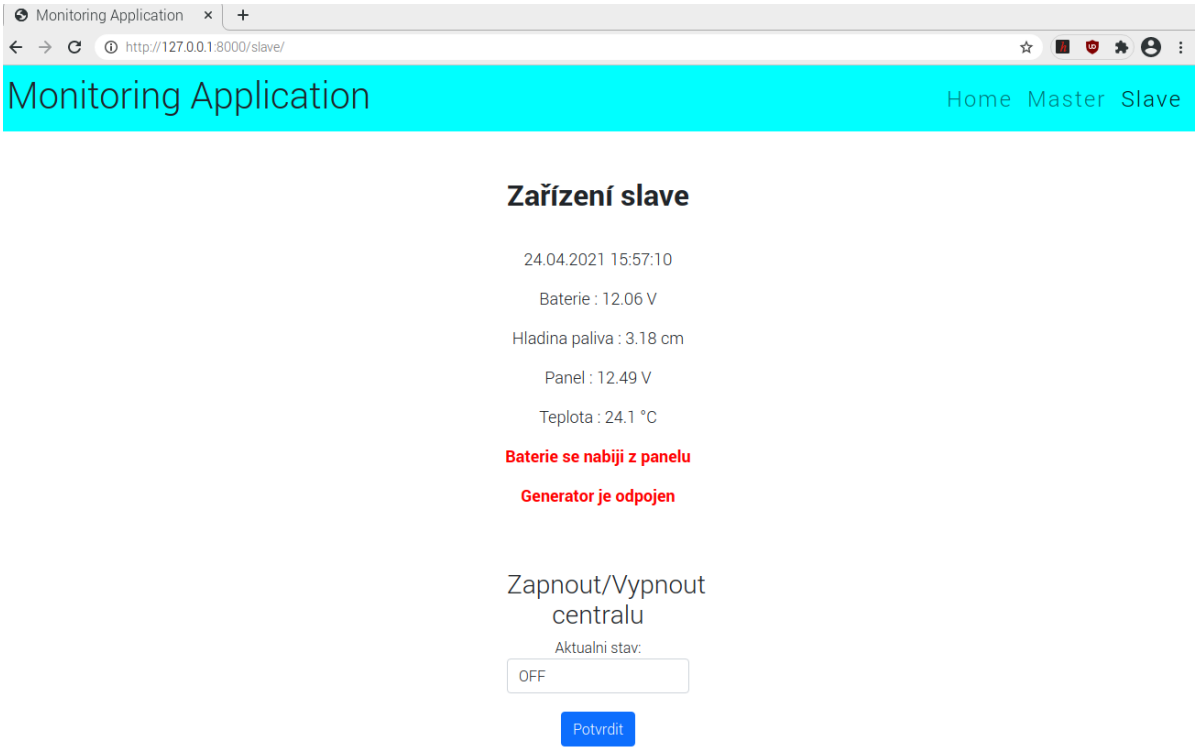


Obrázek 4.2: Hlavní stránka aplikace



Obrázek 4.3: Úprava profilů

Stránky věnované informacím o zařízeních master a slave jsou velmi podobné, protože obě tyto zařízení využívají stejné zdroje a vzhled stránky je nastaven podle jednoduchého konfiguračního souboru *config.ini*. V tomto souboru jsou přednastaveny všechny proměnné do podoby prázdného listu a následně jsou do nich postupně vkládány všechny získané hodnoty ať už ze zařízení master nebo slave. Následně se tyto hodnoty zobrazí na webovém rozhraní. Na těchto stránkách dále uživatel může vypnout nebo zapnout relé, které ovládá elektrocentrálu. V momentě, kdy se tak stane, uživatel přebírá kontrolu nad programem. To znamená, že elektrocentrálu může vypnout pouze on, nebo ji program vypne sám jakmile dojde k překročení nízke hladiny paliva v nádrži centrály. V této chvíli opět přebírá kontrolu systém. Dalším způsobem jak vrátit kontrolu systému je přepnutí profilu.



Obrázek 4.4: Informace o zařízení slave

```
#konfiguracni soubor pro nastaveni templatu
#obe zarizeni mohou mit stejne, ale i odlisne zdroje

[master]
datetime=[]
batteries = []
generators= []
fuellevels= []
panels= []
temperature=[]
relays = []
infos = []

[slave]
datetime=[]
batteries = []
generators= []
fuellevels= []
panels= []
temperature=[]
relays = []
infos = []
```

Obrázek 4.5: Konfigurační soubor *config.ini*

Kapitola 5

Testování

Poslední částí této práce je samotné testování systému. Po spuštění systému okamžitě začne monitoring všech zdrojů. Systém se také hned začne řídit podle aktivního profilu. Systém porovnává naměřené hodnoty s těmi uloženými v profilu.

```
[profile1]
name = Normalni
active = 0
minvoltage = 12.3
minfuellevel = 12
id = 1
timefrom = 10:00:00
timeto = 15:00:00

[profile2]
name = Usporny
active = 1
minvoltage = 12.5
minfuellevel = 15
id = 0
timefrom = 09:30:00
timeto = 17:30:00
```

Obrázek 5.1: Profily

Ihned po prvním monitoringu se odešle zpráva s naměřenými daty na zařízení master. Zařízení master zprávu zachytí a odpoví. Data, která byla zachycena, se pak zobrazí na webovém rozhraní. Takto systém funguje v ideálním případě, kdy je baterie nabitá a nebo se nabíjí ze solárního panelu. Zařízení si navzájem vyměňují informace a vše běží jak má.

V momentě, kdy se uživatel rozhodne změnit aktivní profil, odešle se zvolený profil na zařízení slave. Zařízení tuto zprávu zachytí a přepne právě aktivní profil na profil z přijaté zprávy. Systém se hned začne řídit podle nového aktivního profilu.

```
cas: 2021-04-24 13:19:30,407 | INFO | modul: main | zprava: Start aplikace...
cas: 2021-04-24 13:19:30,417 | INFO | modul: fuelLevelTesting | zprava: Hladina paliva je 3.17, zjisteno na pinu 27
cas: 2021-04-24 13:19:30,848 | INFO | modul: voltageChecking | zprava: Namereno napeti 12.24 na kanale 1
cas: 2021-04-24 13:19:31,328 | INFO | modul: voltageChecking | zprava: Namereno napeti 0.0 na kanale 2
cas: 2021-04-24 13:19:32,799 | INFO | modul: voltageChecking | zprava: Namereno napeti 12.49 na kanale 8
cas: 2021-04-24 13:19:32,810 | INFO | modul: profileExecutor | zprava: Baterie se nabiji z panelu. Napeti na baterii: 12.24
cas: 2021-04-24 13:19:33,370 | INFO | modul: tempAndHumTesting | zprava: Namerena teplota 24.2 na pinu 4
cas: 2021-04-24 13:19:33,373 | INFO | modul: main | zprava: Poslana zprava:  b'{"dateTime": "24.04.2021 13:19:30", "fuelLevels": [3.17],
cas: 2021-04-24 13:19:33,373 | INFO | modul: main | zprava: Cekani na odpoved
cas: 2021-04-24 13:19:33,577 | INFO | modul: main | zprava: Prijata zprava:  'OK'
```

Obrázek 5.2: Start aplikace - výpis z logu

```
cas: 2021-04-24 13:19:43,662 | INFO | modul: main | zprava: Cekam na odpoved.
cas: 2021-04-24 13:19:43,663 | INFO | modul: main | zprava: Prijato 280 bytu od ('192.168.0.101', 43994)
cas: 2021-04-24 13:19:43,664 | INFO | modul: main | zprava: Poslano 2 bytu zpatky na ('192.168.0.101', 43994)
cas: 2021-04-24 13:19:43,664 | INFO | modul: main | zprava: Prijata zprava: {"dateTime": "24.04.2021 13:19:33", "fuelLevels": [3.2],
cas: 2021-04-24 13:19:43,667 | INFO | modul: main | zprava: Profily v poradku
cas: 2021-04-24 13:19:43,669 | INFO | modul: main | zprava: Rele v poradku
cas: 2021-04-24 13:19:46,200 | INFO | modul: tempAndHumidity | zprava: Namerena teplota 24.8 na pinu 2
cas: 2021-04-24 13:19:46,201 | INFO | modul: main | zprava: Cekam na odpoved.
```

Obrázek 5.3: Příjem zprávy - výpis z logu

```
cas: 2021-04-24 13:39:11,369 | INFO | modul: main | zprava: Cekani na odpoved
cas: 2021-04-24 13:39:11,370 | INFO | modul: main | zprava: Prijata zprava:  '{"name": "Usporny", "active": "1",
cas: 2021-04-24 13:39:15,384 | INFO | modul: fuelLevelTesting | zprava: Hladina paliva je 3.62, zjisteno na pinu
cas: 2021-04-24 13:39:15,864 | INFO | modul: voltageChecking | zprava: Namereno napeti 12.35 na kanale 1
cas: 2021-04-24 13:39:17,820 | INFO | modul: voltageChecking | zprava: Namereno napeti 12.49 na kanale 8
cas: 2021-04-24 13:39:17,824 | INFO | modul: profileExecutor | zprava: Baterie se nabiji z panelu. Napeti na bate
```

Obrázek 5.4: Příjem profilu - výpis z logu

Když se uživatel rozhodne zapnout centrálu pro dobití baterie, odelše se zpráva na zařízení slave, které centrálu zapne.

```
cas: 2021-04-27 08:17:29,409 | INFO | modul: main | zprava: Cekani na odpoved
cas: 2021-04-27 08:17:29,410 | INFO | modul: main | zprava: Prijata zprava:  '1'
cas: 2021-04-27 08:17:29,411 | INFO | modul: main | zprava: Centrala zapnuta uzivatelem.
cas: 2021-04-27 08:17:33,441 | INFO | modul: fuelLevelTesting | zprava: Hladina paliva je 3.16, zjisteno na pinu 27
```

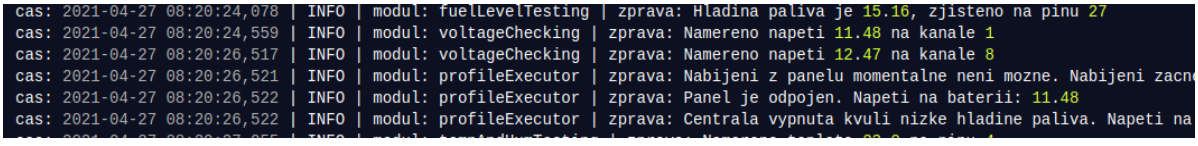
Obrázek 5.5: Zapnutí centrály - výpis z logu

Uživatel sice zasáhl do dění v systému, ale to neznamená, že systém nemůže dále monitorovat zdroje. V momentě, kdy je centrála v provozu a dojde k vyčerpání značného množství paliva, systém centrálu vypne, aby nedošlo k jejímu poškození. Systém umí centrálu nejen nouzově vypnout, ale umí ji i zapnout. A to v momentě, kdy se napětí baterie blíží hranici vybití. Při testování byla bohužel baterie nabitá, proto tato situace nešla otestovat.

V případě nenadálého výpadku nebo odpojení API dojde k zaznamenání chybové hlášky do logu. Tato zpráva se následně promítne i do webového rozhraní. V momentě, kdy dojde ke ztrátě spojení se zařízením slave, se chyba opět zapíše do logu a zařízení master stále čeká na zprávu. Jakmile se stane, že některý ze senzorů neodpovídá, dojde k zapsání chyby do logu a následnému ukončení celé aplikace.

Výsledek testu se dá hodnotit kladně, systém je plně samostatný a zároveň dokáže reagovat na uživatelovy zásahy.

Obrázek 5.6: Zapnutí centrály



Obrázek 5.7: Nouzové vypnutí centrály - výpis z logu

Obrázek 5.8: Nouzové vypnutí centrály

```
cas: 2021-04-24 17:13:59,092 | INFO | modul: main | zprava: Rele v poradku
cas: 2021-04-24 15:14:00,327 | ERROR | modul: views | zprava: Spojeni s API nebylo navazano
cas: 2021-04-24 17:14:11,764 | INFO | modul: tempAndHumidity | zprava: Namerena teplota 24.6 na pinu 2
```

Obrázek 5.9: Nedostupné API - výpis z logu

Obrázek 5.10: Nedostupné API

```
cas: 2021-04-27 08:12:00,685 | INFO | modul: main | zprava: Cekam na odpoved.
cas: 2021-04-27 08:12:05,688 | ERROR | modul: main | zprava: Zadna zprava nebyla prijata. Zkontrolujte zarizeni Slave.
cas: 2021-04-27 08:12:08,234 | INFO | modul: tempAndHumidity | zprava: Namerena teplota 24.9 na pinu 2
cas: 2021-04-27 08:12:08,237 | INFO | modul: main | zprava: Cekam na odpoved.
```

Obrázek 5.11: Nedostupné zařízení slave - výpis z logu

```
cas: 2021-04-27 08:47:58,508 | ERROR | modul: tempAndHumTesting | zprava: Nebyla namerena zadna hodnota na teplomeru.
Traceback (most recent call last):
  File "/home/pi/BC/tempAndHumTesting.py", line 16, in getTempInfo
    log.logger.info('Namerena teplota {} na pinu {}'.format(round(temp,2), pin))
TypeError: type NoneType doesn't define __round__ method
```

Obrázek 5.12: Nedostupný senzor - výpis z logu

Kapitola 6

Závěr

Cílem práce bylo navrhnout systém pro správu záložních zdrojů napájení, který se sám dokáže přizpůsobovat určitým situacím. Systém monitoruje jednotlivé zdroje a předává získané informace uživateli formou webového rozhraní. Systém dokáže reagovat na určité situace a přizpůsobovat se jim. Uživatel sám může kdykoliv zasáhnout a ovlivnit běh systému.

Přesto, že tento vytvořený systém je plně funkční, nabízí se hned několik možností na jeho další rozšíření. Protože se v systému již pracuje s časem (od kdy do kdy používat nabíjení pomocí solárních panelů), nabízí se jako jedna z možností vytvoření kalendáře. Zde si uživatel bude moct naplánovat spuštění jednotlivých profilů, nebo např. jen spuštění elektrocentrály. Toto by bylo užitečné např. v případě, kdy by uživatel chtěl mít na každý víkend plně nabité baterie. Dalším možným přídavkem by mohla být meteorologická stanice. V kombinaci se solárními panely nebo větrnými turbínami, které by nabíjely baterie, by se jednalo o vskutku dobré rozšíření. Program by mohl sám rozhodovat, zda zapnout nebo vypnout nabíjení pomoci výše zmíněných zdrojů, podle počasí.

Literatura

1. *oEnergetice.cz - denní zpravodajství z energetiky: Záložní zdroje elektrické energie - 1.díl: Úvod do problematiky*. 2015. Dostupné také z: `https://oenergetice.cz/technologie/zalozni-zdroje-elektricke-energie-1-dil-uvod-do-problematiky`.
2. *BCE.cz. Fotovoltaická elektrárna na klíč pro váš dům: Fotovoltaické panely – 3 základní typy* [online] [cit. 2021-04-14]. Dostupné z: `https://www.bce.cz/3-typy-fotovolatickych-panelu/`.
3. *Raspberry Pi expansion boards and accessories from AB Electronics UK: ADC Pi - ADC converter for the Raspberry Pi*. Dostupné také z: `https://www.abelectronics.co.uk/p/69/adc-pi-raspberry-pi-analogue-to-digital-converter`.
4. *H A D E X , spol. s r.o. [online]: Regulátor MPPT 9-12-24V 5A, modul MPPT-V08A s IO CN3722* [online] [cit. 2021-04-14]. Dostupné z: `https://www.hadex.cz/m464-regulator-mppt-9-12-24v-5a-modul-mppt-v08a-s-io-cn3722/`.
5. BRADBURY, Alex; EVERARD, Ben; WINDER, Russe. *Learning Python with Raspberry Pi*. John Wiley & Sons, 2014.
6. UPTON, Eben; HALFACREE, Gareth. *Raspberry Pi: uživatelská příručka*. Brno: Computer Press, 2013.
7. *FrontPage - Raspbian: FrontPage - Raspbian*. Dostupné také z: `http://www.raspbian.org/`.
8. *NOOBS - Raspberry Pi návody. Raspberry Pi návody*. 2021. Dostupné také z: `https://raspberry.pavelrampas.cz/noobs/`.
9. GARETH, Halfacree. *THE OFFICIAL Raspberry Pi Beginner's Guide, How to use your new computer*. Raspberry Pi PRESS, [n.d.].
10. *Raspberry Pi 4 Computer: Raspberry Pi 4 Computer Model B*. Raspberry Pi Trading Ltd, 2019. Dostupné také z: `www.raspberrypi.org`.
11. *ProgrammerSought* [online] [cit. 2021-04-14]. Dostupné z: `https://www.programmersought.com/article/82985565233/`.
12. LUTZ, Mark. *Learning Python. 4th ed*. Beijing: O'Reilly, 2009.

13. *Django: The Web framework for perfectionists with deadlines* [online] [cit. 2021-04-20]. Dostupné z: `https://www.djangoproject.com/`.

14. *Django – Wikipedie* [online] [cit. 2021-04-20]. Dostupné z: `https://cs.wikipedia.org/wiki/Django`.

15. HOLOVATY, Adrian; KAPLAN-MOSS, Jacob; GILMORE, Jason. *The definitive guide to Django: Web development done right.* New York: Springer-Verlag, 2008.